Autonomous Vehicle

Ahmed Mady *46-0798* | Ahmed Yasser *46-12951* | Mohamed Magdy *46-3493* | Mostafa Kashaf *46-5456* | Omar Mohsen *46-5023*

***Abstract*—Autonomous vehicles (AVs) represent a revolutionary technological advancement in transportation. As these vehicles become increasingly prevalent, it is imperative to understand the state of research surrounding their implementation and impact. In this report, we simulated the P-Controller and the Stanley Controller on Gazebo Software using ROS to be able to control the longitudinal and lateral motions of the vehicle. Then, we used these controllers on a real car to accurately control its motion, both in moving in a straight line or obstacle avoidance.**

***Index Terms*—autonomous, P-controller, Stanley Controller, Gazebo, ROS**

I. Introduction

In the realm of transportation, a revolutionary breakthrough has emerged, promising to reshape the way we perceive and interact with the world around us. Autonomous vehicles, also known as self-driving cars, represent a remarkable fusion of cutting-edge technology and transportation engineering. With the potential to redefine mobility as we know it, these vehicles are poised to create a seismic shift in various sectors, ranging from personal transportation and logistics to public transit and urban planning.

Autonomous vehicles have captivated the imagination of researchers, policymakers, and the public alike, as they offer a tantalizing glimpse into a future where human drivers are no longer a prerequisite for transportation. Leveraging advanced sensors, artificial intelligence, and real-time data processing, these vehicles possess the capability to navigate and operate independently, exhibiting an impressive level of accuracy, efficiency, and safety.

Throughout this report, we will investigate the underlying technology that empowers autonomous vehicles, including sensor systems, artificial intelligence algorithms, and communication networks. We will examine the various levels of autonomy, ranging from driver-assistance systems to fully autonomous vehicles, highlighting the technological milestones achieved thus far.

As we embark on this journey into the realm of autonomous vehicles, it is crucial to navigate the discourse surrounding their development with an informed perspective. By thoroughly examining the technology, applications, and societal implications, we aim to contribute to a robust understanding of autonomous vehicles, enabling stakeholders and policymakers to make informed decisions about the future of transportation.

In this report, we implemented an autonomous vehicle using a raspberry pi 4b integrated with a couple of sensors. The targets were to make it move in a straight line and to make it follow a certain path to avoid obstacles. We also implemented this autonomous vehicle on simulation in Gazebo which uses Robot Operating System (ROS) to make the nodes communicate with each other and move the vehicle in the desired paths.

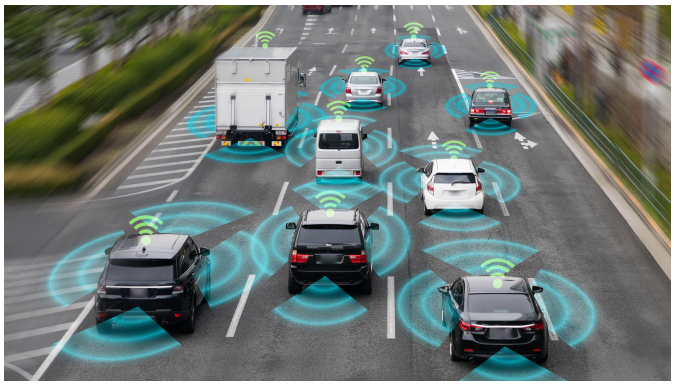

Fig. 1. Autonomous Vehicles

II. Literature Review

A. Technological Advancements

The technological advancements in autonomous vehicles have been a primary focus of research. Studies have explored various aspects, including perception systems, sensor fusion, decision-making algorithms, and communication infrastructure. Zhang et al. (2019) ([1]) highlighted the significance of machine learning and artificial intelligence techniques in enabling AVs to interpret complex traffic scenarios. Additionally, research by Li et al. (2021) ([2]) emphasized the importance of robust localization and mapping technologies for accurate navigation in real-world environments.

B. Safety Considerations

Ensuring the safety of autonomous vehicles has been a major concern. Research has investigated the reliability of perception systems, response to unpredictable events, and the ability to handle various weather conditions. Levinson et al. (2020) ([3]) discussed the challenges of designing safety systems that can detect and respond to rare and anomalous events effectively. Moreover, Liang et al. (2022)([4]) explored the potential of reinforcement learning algorithms to enhance the safety of AVs by training them in simulated environments.

C. Ethical Implications

The deployment of autonomous vehicles raises significant ethical considerations. Research has examined topics such

as decision-making algorithms in critical situations, liability issues, and the impact on employment. Lin et al. (2019)([5]) delved into the ethical challenges faced by AVs in situations where a choice between preserving passengers' lives and those of pedestrians arises. Moreover, Anderson et al. (2021)([6]) highlighted the need for clear guidelines and legal frameworks to address liability concerns when accidents involving autonomous vehicles occur.

D. Societal Impacts

The widespread adoption of autonomous vehicles is expected to have profound societal impacts. Research has explored potential changes in urban infrastructure, commuting patterns, environmental sustainability, and accessibility. Hong et al. (2020)([7]) examined the potential for reducing traffic congestion and emissions through the deployment of autonomous ride-sharing services. Furthermore, Hamidi et al. (2021)([8]) investigated the impact of AVs on accessibility for marginalized populations and the potential for reducing transportation inequalities.

III. Methodology

A. Simulation

1) Linux Ubuntu: Using Linux Ubuntu as the operating system for working with ROS Noetic provides several advantages for developers and researchers in the field of robotics including easy integration, a large user community, stability, extensive software ecosystem, a developer-friendly environment, and customization options. These benefits make Ubuntu a popular choice for ROS development and contribute to a streamlined and efficient workflow for robotics projects.

2) ROS: ROS, or Robot Operating System, is an open-source framework widely used in robotics research and development. It provides a flexible and modular platform for building robotic systems by offering a collection of software libraries, tools, and conventions. ROS follows a distributed architecture, allowing different components of a robotic system to communicate and share information seamlessly. It provides a wide range of functionalities, including message passing, inter-process communication, and hardware abstraction, which simplify the development and integration of complex robotic systems. Additionally, ROS fosters code reusability and collaboration through its package management system, enabling developers to share their work and leverage existing libraries and algorithms. With its extensive community support, ROS has become a standard in the robotics community, empowering researchers and developers to focus on higher-level tasks and accelerating the advancement of robotics technology.

3) Gazebo: Gazebo is an open-source, multi-robot simulator widely used in the field of robotics research and development. It provides a realistic and dynamic environment for simulating robots, sensors, and their interactions with the surroundings. Gazebo offers a wide range of features, including physics simulation, sensor emulation, and visualization capabilities, allowing researchers and developers to test and validate their algorithms and control strategies in a virtual environment before deploying them on real robots. The flexible architecture of Gazebo enables the creation of complex scenarios and the integration of various sensors and actuators, making it a versatile tool for simulating different types of robots and environments. With its extensive community support and rich set of plugins and models, Gazebo has become a cornerstone in the robotics community, fostering innovation and accelerating the development of autonomous systems.

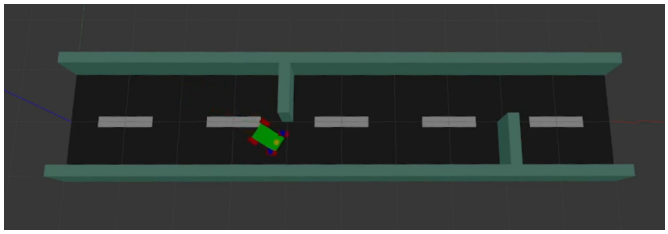

Fig. 2. Vehicle Model in Gazebo Software

We started by implementing an open loop node which only takes the speed as an input to test the model, taken from an open source. Then we implemented a tele-operation node, so it would be tested in the simulation before trying it on the hardware. Thirdly, speed control and lateral control were implemented on the model. The speed control was made using a P-Controller, because it was simple and stable. The lateral control was made using the Stanley controller, which employs an advanced concept called "heading control", meaning that the vehicle steers towards a specific heading angle that is derived from the desired path trajectory, rather than trying to directly match its position with the path. This approach provides robustness in dealing with noise and uncertainties in sensor measurements.

Next, We designed the map to have the required track with a couple obstacles for the vehicle to make maneuvers in (Figure 2), and we also made the planning module node which decides when will the vehicle shift lanes and change speeds.

Finally, we made a localization node to add white Gaussian noise to the model, then remove it using Kalman Filter to test it before implementing it on the hardware. It showed promising results, but we had to tune the Stanley controller gain for better performance.

B. Hardware

To build an autonomous car with two sensors—an Inertial Measurement Unit (IMU) and a Hall effect sensor—as well as two actuators—a DC motor and a servo motor—we followed a step-by-step process. The car, which was purchased in a scale of 1:18, was equipped with these components. It was driven by a Raspberry Pi 4 Model 4B+ and controlled using the Robot Operating System (ROS).

1) Raspberry Pi 4 Model 4B+: The Raspberry Pi 4 Model 4B+ was chosen as the main computing platform for the autonomous car. It features a quad-core ARM Cortex-A72 processor, 4GB of RAM, and provides ample computational power for running the necessary algorithms and controlling

the actuators. Additionally, it has built-in Wi-Fi and Bluetooth capabilities, enabling easy communication with other devices.

2) Sensor Installation: The initial sensor utilized was an IMU, positioned on the car to measure the vehicle's yaw. This information was crucial for estimating the car's position. The IMU was securely mounted and connected to the Raspberry Pi, enabling real-time data acquisition. The second sensor used to read the velocity and distance moved by the car was a Hall effect sensor. A magnet was placed on one of the wheels, and the Hall effect sensor was positioned to detect the changes in the magnetic field as the wheel rotated. This data was then transmitted to the Raspberry Pi for further processing.

3) Actuator Installation: Two actuators were integrated into the car's system. Firstly, a DC motor was connected to the back wheels of the car to provide propulsion. This motor was controlled through an H-bridge with a gearbox, allowing for variable speed and direction control. The H-bridge was connected to the GPIO pins of the Raspberry Pi, enabling precise control over the motor's movements. The second actuator was a servo motor, which was responsible for steering the car. It was connected to the front wheels, enabling changes in direction. The servo motor was also linked to the Raspberry Pi and controlled using PWM signals.

4) Software Setup and ROS Nodes: The software of the autonomous car was developed using ROS, which provided a robust framework for communication among various modules. Three nodes were created to facilitate different functionalities. The longitudinal node publishes the driving velocity and distance while subscribing to the longitudinal velocity information. This node plays a crucial role in maintaining a straight line trajectory using the Stanley controller. The lateral node subscribes to velocity, lateral distance, and distance information. It processes this data to execute lane changes and obstacle avoidance, ensuring the safe lateral movement of the car. The planning node subscribes to velocity, lateral distance, and distance information. It uses these inputs to calculate lane distance and speed, providing the necessary information for executing the planned path.

5) Autonomous Movements: The autonomous car was capable of performing two movements: straight-line driving using the Stanley controller, and simple path planning for changing lanes and avoiding obstacles using if conditions. When operating in a straight line, the longitudinal node publishes the desired velocity and distance while receiving feedback on the longitudinal velocity. The Stanley controller, implemented in the software, adjusts the car's movements based on the yaw measurements from the IMU, allowing for precise navigation. For more complex movements, such as changing lanes and avoiding obstacles, the lateral and planning nodes come into play. These nodes process sensor data and make decisions based on predetermined conditions, ensuring that the car safely maneuvers through its environment.

By adhering to this methodology, the autonomous car was equipped successfully with the required sensors and actuators.

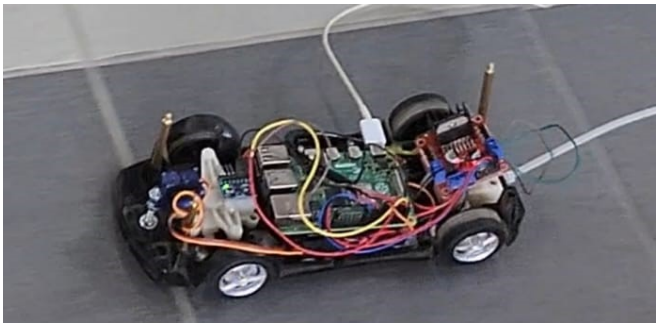

Fig. 3. Implemented Autonomous Car

Lane_init	Heading	V_des	Lane_des	X_err	Y_err	Yaw_err	Vel_err	Time
0	0	1	1	0.0394	3.6e-5	0.00015	0.0061	13.12
0	0	1	5	0.0307	4.2e-5	7e-5	0.0158	13.53
0	0	3	5	1.95	1.5e-4	6.6e-5	0.0345	8.31
2	0	1	5	0.04	2e-5	0.00015	0.017	13.05
2	0	1	-3	0.052	3e-5	1.9e-5	1.8e-3	13.14
-6	0	1	-1	0.051	5e-5	10e-6	0.0115	13.11
0	1	1	1	0.054	5.4e-5	6e-5	0.0146	12.92
0	2	1	1	0.06	5.4e-5	1.8e-4	0.0141	13.11
0	3	1	1	0.05	2e-5	2.7e-5	0.0133	13.93
0	4	1	1	0.06	10e-5	5.4e-5	0.0167	13.52
2	3	1	-4	0.042	9e-5	10e-4	0.0143	14.15
-2	2	2	5	0.69	10e-4	0.0002	0.0255	9.22
-1	3	1	4	0.06	5e-4	0.00018	0.0175	14.52

Fig. 4. Different Tests

The Raspberry Pi 4 Model 4B+ served as the central control unit, running ROS and facilitating seamless communication between the various nodes. The car's movements were precisely controlled and executed, thanks to the integration of the IMU, Hall effect sensor, DC motor, and servo motor, all of which contributed to its autonomy and intelligence (Figure 3).

IV. Results

A. Simulation Results

The P-Controller, Stanley Controller, and Kalman filter results were very promising where the error was of a small value, which made the vehicle move smoothly in the simulation. We plotted the graphs of the different car states and the desired speed and lane vs the actual speed and lane to test the effectiveness of the controllers and the Kalman filter.

1) Controller Tests: We ran a lot of tests by changing the initial conditions (position and heading) to find out the response of both controllers (Figure 4). We tested many different headings and lanes and the car reached the desired lanes with minimal error.

2) Filtered X: Since there were not much maneuvers in the X- direction, the plot seems very smooth as well as the filtered direction from the Gaussian noise (Figure 5)

3) Filtered Y: The y axis is the lane axis, that's why the plots of both will be the same (Figure 6). The Gaussian noise appeared here more, because there are many maneuvers taken. The steering angle keeps oscillating making these sinusoidal graphs. The steering oscillates, because that's is how the Stanley controller operates. It follows a look-ahead point at every time step to reach its destination. The increase in speed

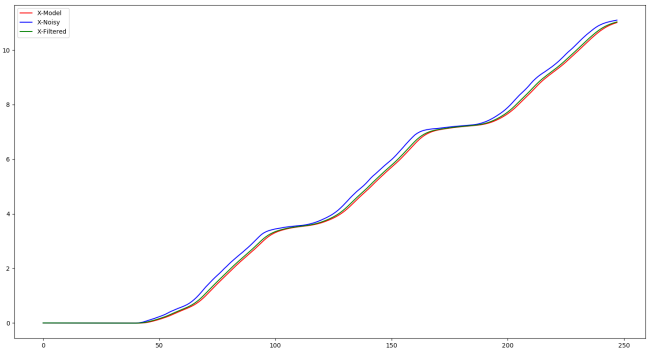

Fig. 5. X-Plot

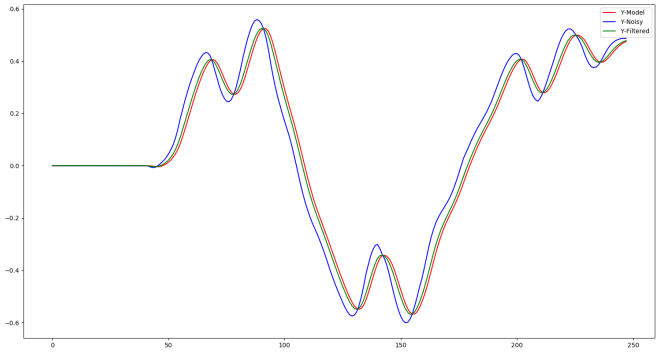

Fig. 6. Y-Plot

also increases the oscillations of the car. To decrease these oscillations the Stanley gain is adjusted to the optimal for that specific speed. Every speed has its own gain.

4) Filtered Theta: As mentioned in the previous section, the oscillations are a lot in the Stanley Controller until the car stabilizes. As seen in this plot (Figure 7), these oscillations were worse, until the Stanley gain was adjusted which resulted in these better plots.

5) Speed Plot: Figure 8 plots the desired speed vs the actual speed of the model car. It starts from rest, then reaches first lane while decreasing its speed a little for safe maneuver. For the second lane switching, the desired speed was decrease on purpose for safe maneuver, and the controller also decreases the speed, that's why the actual speed plot is much lower than the desired speed. It repeats for the next lane changing.

6) Lane Plot: As mentioned before, the lane plot is exactly the same as the y plot, because the lane is the y axis. At the beginning, the car reaches the first lane, that's why there is a little maneuver, then it starts the lane changing sequence.

B. Hardware Results

The same nodes were implemented on the hardware except for the localization node, because our sensor does not have a

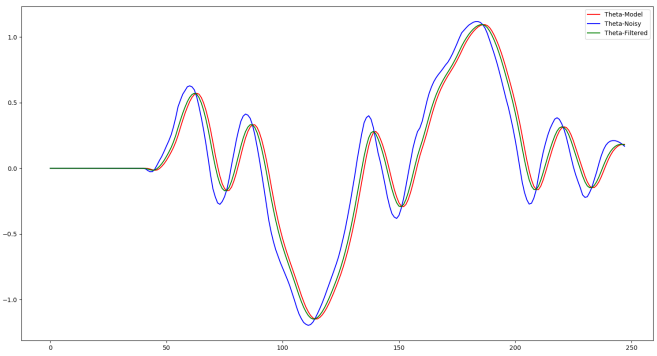

Fig. 7. Theta-Plot

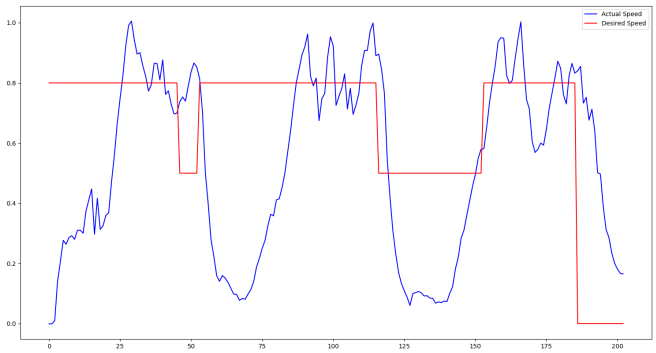

Fig. 8. Speed-Plot

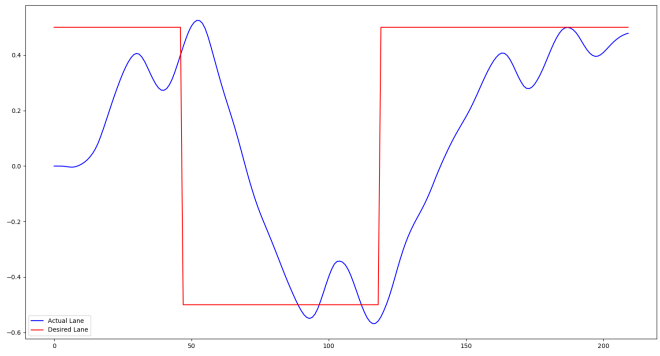

Fig. 9. Lane-Plot

Fig. 10. Case 1: Straight Line

lot of noise, and its performance is very good for the project requirement. It took us days of tuning to reach the required performance. We kept changing the Stanley gains for every speed, until we reached the optimal gain for the car's top speed. We made two cases. The first case was for the car to move in a straight line for a distance of 10 meters. The car's performance was exactly like the plots above, where it kept oscillating around the desired lane while maintaining the lane at its center. The car oscillated a lot, because the speed was high (almost 1m/s) =, which was a great experiment. The second case was that the car would change lanes to avoid hitting the obstacles. It performed well at the same high speeds, since we let it change lanes more than a meter away from each obstacles to avoid hitting them. This figure (10) shows the actual car at the end of the 10-meter track maintaining the lane while oscillating around the lane exactly.

This figure (11) shows the actual car at the beginning of the obstacle avoidance track, while changing lanes to the desired lane away from the obstacle.

V. Conclusion

In conclusion, we implemented the ROS nodes in the simulation on Gazebo software to test the performance of the longitudinal and lateral controllers controllers before implementing them on the hardware to ensure smooth performance without errors. We then too these ROS nodes on the Raspberry Pi and made some adjustments like adding the ports that are responsible for the operation of the actuators. Then we made a lot of tunings until we reached the desired gains that resulted in smooth and accurate performance. We finally launched all

Fig. 11. Case 2: Obstacle Avoidance

the used nodes together in a launch file for the Raspberry Pi to constantly run all nodes simultaneously as this is the most powerful use for ROS. Finally, the car managed to finish the 2 required race tracks in a fast navigation and accurate positioning.

References

[1] Zhang, S., Yao, L., Sun, A.,Tay, Y. (2019). Deep learning based recommender system: A survey and new perspectives. ACM computing surveys (CSUR), 52(1), 1-38.

[2] Tseng, K. K., Li, J., Chang, Y., Yung, K. L., Chan, C. Y.,Hsu, C. Y. (2021). A new architecture for simultaneous localization and mapping: an application of a planetary rover. Enterprise Information Systems, 15(8), 1162-1178.

[3] Ji, A., Levinson, D. (2020). A review of game theory models of lane changing. Transportmetrica A: transport science, 16(3), 1628-1647.

[4] Wang, X., Wang, S., Liang, X., Zhao, D., Huang, J., Xu, X., ... Miao, Q. (2022). Deep reinforcement learning: a survey. IEEE Transactions on Neural Networks and Learning Systems.

[5] Wang, X., Tajvidi, M., Lin, X., Hajli, N. (2020). Towards an ethical and trustworthy social commerce community for brand value co-creation: A trust-commitment perspective. Journal of Business Ethics, 167, 137-152.

[6] Murray, A., Kuban, S., Josefy, M., Anderson, J. (2021). Contracting in the smart era: The implications of blockchain and decentralized autonomous organizations for contracting and corporate governance. Academy of Management Perspectives, 35(4), 622-641.

[7] Lv, M., Hong, Z., Chen, L., Chen, T., Zhu, T., Ji, S. (2020). Temporal multi-graph convolutional network for traffic flow prediction. IEEE Transactions on Intelligent Transportation Systems, 22(6), 3337-3348.

[8] Zandiatashbar, A., Hamidi, S. (2021). Transportation amenities and high-tech firm location: An empirical study of high-tech clusters. Transportation Research Record, 2675(12), 820-831.